PTO/SB/21 (09-04)
Approved for use through 07/31/2006. OMB 0651-0031
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE


# TRANSMITTAL FORM

*(to be used for all correspondence after initial filing)*

Total Number of Pages in This Submission: 18

| | |
|---|---|
| Application Number | 09/928,565 |
| Filing Date | 02/28/2001 |
| First Named Inventor | Elliot K. Kolodner |
| Art Unit | 2195 |
| Examiner Name | Nilesh R. Shah |
| Attorney Docket Number | GB920000101US1 |

## ENCLOSURES *(Check all that apply)*

- [x] Fee Transmittal Form
  - [ ] Fee Attached
- [ ] Amendment/Reply
  - [ ] After Final
  - [ ] Affidavits/declaration(s)
- [ ] Extension of Time Request
- [ ] Express Abandonment Request
- [ ] Information Disclosure Statement
- [ ] Certified Copy of Priority Document(s)
- [ ] Reply to Missing Parts/ Incomplete Application
  - [ ] Reply to Missing Parts under 37 CFR 1.52 or 1.53
- [ ] Drawing(s)
- [ ] Licensing-related Papers
- [ ] Petition
- [ ] Petition to Convert to a Provisional Application
- [ ] Power of Attorney, Revocation Change of Correspondence Address
- [ ] Terminal Disclaimer
- [ ] Request for Refund
- [ ] CD, Number of CD(s) _____
  - [ ] Landscape Table on CD
- [ ] After Allowance Communication to TC
- [ ] Appeal Communication to Board of Appeals and Interferences
- [x] Appeal Communication to TC **(Appeal Notice, Brief, Reply Brief)**
- [ ] Proprietary Information
- [ ] Status Letter
- [x] Other Enclosure(s) (please Identify below):

Postcard

Remarks

## SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT

| | | | |
|---|---|---|---|
| Firm Name | Holland + Knight LLP | | |
| Signature | Michael J. Buchenhorner | | |
| Printed name | Michael J. Buchenhorner | | |
| Date | Jan. 4, 2006 | Reg. No. | 33,162 |

## CERTIFICATE OF TRANSMISSION/MAILING

I hereby certify that this correspondence is being facsimile transmitted to the USPTO or deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on the date shown below:

| | | | |
|---|---|---|---|
| Signature | Michael J. Buchenhorner | | |
| Typed or printed name | Michael J. Buchenhorner | Date | Jan. 4, 2006 |

PTO/SB/17 (12-04v2)
Approved for use through 07/31/2006. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE


*Effective on 12/08/2004.*
*Fees pursuant to the Consolidated Appropriations Act, 2005 (H.R. 4818).*

# FEE TRANSMITTAL
## For FY 2005

☐ Applicant claims small entity status. See 37 CFR 1.27

| TOTAL AMOUNT OF PAYMENT | ($) | 500.00 |
|---|---|---|

| Complete if Known | |
|---|---|
| Application Number | 09/928,565 |
| Filing Date | 02/28/2001 |
| First Named Inventor | Elliot K. Kolodner |
| Examiner Name | Nilesh R. Shah |
| Art Unit | 2195 |
| Attorney Docket No. | GB920000101US1 |

**METHOD OF PAYMENT** (check all that apply)

☐ Check ☐ Credit Card ☐ Money Order ☐ None ☐ Other (please identify): _____

☑ Deposit Account Deposit Account Number: 50-0510 Deposit Account Name: IBM T.J.Thomas Rsrch Cnt

For the above-identified deposit account, the Director is hereby authorized to: (check all that apply)

☑ Charge fee(s) indicated below ☐ Charge fee(s) indicated below, except for the filing fee

☑ Charge any additional fee(s) or underpayments of fee(s) under 37 CFR 1.16 and 1.17 ☑ Credit any overpayments

**WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.**

**FEE CALCULATION**

**1. BASIC FILING, SEARCH, AND EXAMINATION FEES**

| | FILING FEES | | SEARCH FEES | | EXAMINATION FEES | | |
|---|---|---|---|---|---|---|---|
| | | Small Entity | | Small Entity | | Small Entity | |
| Application Type | Fee ($) | Fee ($) | Fee ($) | Fee ($) | Fee ($) | Fee ($) | Fees Paid ($) |
| Utility | 300 | 150 | 500 | 250 | 200 | 100 | |
| Design | 200 | 100 | 100 | 50 | 130 | 65 | |
| Plant | 200 | 100 | 300 | 150 | 160 | 80 | |
| Reissue | 300 | 150 | 500 | 250 | 600 | 300 | |
| Provisional | 200 | 100 | 0 | 0 | 0 | 0 | |

**2. EXCESS CLAIM FEES**

| Fee Description | Fee ($) | Small Entity Fee ($) |
|---|---|---|
| Each claim over 20 (including Reissues) | 50 | 25 |
| Each independent claim over 3 (including Reissues) | 200 | 100 |
| Multiple dependent claims | 360 | 180 |

| Total Claims | | Extra Claims | | Fee ($) | | Fee Paid ($) |
|---|---|---|---|---|---|---|
| _____ | - 20 or HP = | _____ | x | _____ | = | _____ |

HP = highest number of total claims paid for, if greater than 20.

| Indep. Claims | | Extra Claims | | Fee ($) | | Fee Paid ($) |
|---|---|---|---|---|---|---|
| _____ | - 3 or HP = | _____ | x | _____ | = | _____ |

HP = highest number of independent claims paid for, if greater than 3.

| Multiple Dependent Claims | |
|---|---|
| Fee ($) | Fee Paid ($) |
| _____ | _____ |

**3. APPLICATION SIZE FEE**

If the specification and drawings exceed 100 sheets of paper (excluding electronically filed sequence or computer listings under 37 CFR 1.52(e)), the application size fee due is $250 ($125 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR 1.16(s).

| Total Sheets | | Extra Sheets | | Number of each additional 50 or fraction thereof | | Fee ($) | | Fee Paid ($) |
|---|---|---|---|---|---|---|---|---|
| _____ | - 100 = | _____ | / 50 = | _____ | (round up to a whole number) x | _____ | = | _____ |

**4. OTHER FEE(S)**

| | Fees Paid ($) |
|---|---|
| Non-English Specification, $130 fee (no small entity discount) | |
| Other (e.g., late filing surcharge): Brief on Appeal | 500.00 |

**SUBMITTED BY**

| Signature | Michael J. Buchenhorner | Registration No. (Attorney/Agent) 33,162 | Telephone 305-789-7773 |
|---|---|---|---|
| Name (Print/Type) | Michael J. Buchenhorner | | Date Jan. 4, 2006 |

ZZW AF

Attorney's Docket No.: GB920000101US1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

| | | | |
|---|---|---|---|
| Applicant | : Elliot Karl Kolodner | Art Unit | : 2195 |
| Serial No. | : 09/928,565 | Examiner | : Nilesh R. Shah |
| Filed | : February 28, 2001 | | |
| Title | : Computer System with Heap Reset | | |

**Mail Stop Appeal Brief - Patents**
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

BRIEF ON APPEAL

**(1) Real Party in Interest**

International Business Machines Corporation is the real party in interest.

**(2) Related Appeals and Interferences**

None known.

**(3) Status of Claims**

Claims 1-34 are pending in the case. (See Appendix of Claims), Claims 1-11 and 23-34 were rejected under 35 U.S.C. §101 as being directed to non-statutory subject matter. Claims 1-33 were rejected under §103(a) as having been unpatentable over Wolczko et al. (U.S. Patent 5,900,001, hereafter "Wolczko" ) in view of Aman et al (U.S. Patent 6,694,346, hereafter, "Aman"). All of the pending claims are being appealed.



CERTIFICATE OF MAILING BY FIRST CLASS MAIL

I hereby certify under 37 CFR §1.8(a) that this correspondence is being deposited with the United States Postal Service as first class mail with sufficient postage on the date indicated below and is addressed to the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

January 4, 2006
Date of Deposit

Michael J. Buchenhorner
Signature

Michael J. Buchenhorner
Typed or Printed Name of Person Signing Certificate

**(4) Status of Amendments**

No substantive amendments have been made since the final office action dated August 1, 2005.

**(5) Summary of Claimed Subject Matter**

Claim 1 recites a computer system [specification page 12, line 21, item 10] providing an object-based virtual machine [specification page 12, line 30, item 40] environment for running successive applications, the computer system comprises storage [specification page 12, line 23, item 60], at least a portion of which is logically divided into two or more heaps [page 6, line 1; FIG. 5, items 510 and 520] in which objects can be stored, wherein a first heap [page 6, line 2, FIG. 5, item 520] is reset between successive applications, and a second heap [page 6, line 3, FIG. 5 item 510] persists from one application to the next; a card table [page 6, line 5, FIG. 5, item 536] comprising multiple cards [page 14, col. 1, page 59, lines 3-25], each corresponding to a region [page 6, line 5, page 59, lines 6-9] of said storage, each card in the card table being set to null when the first heap is reset between successive applications [page 6, lines 5-8; page 62, lines 6-12]; logic for marking a card whenever an object in its corresponding storage region is updated [page 6, lines 9-10, page 59, line 22 – page 60, line 2; FIG. 5, item 536]; and logic for detecting possible references from the second heap to the first heap at reset by scanning the cards in the card table corresponding to the second heap, and detecting any cards which have been marked [page 60, lines 3-16]. Claims 2-9 are dependent on claim 1.

Claim 10 relates to a computer system [FIG. 1, item 10] providing an object-based virtual machine environment [FIG. 1, item 40] for running successive applications. The computer system comprises storage [FIG. 1, item 60], at least a portion of which is logically divided into two or more heaps [FIG. 5, item 140] in which objects [FIG. 1, items 145] can be stored, wherein a first heap [FIG. 5, item 520] is reset between successive applications, and a second heap [FIG. 5, item 520] persists from one application to the next. The means for identifying any objects on the first heap which have a finalization method corresponds to the microprocessor 20 interacting with the OS 30, JVM 40, and other components in FIG. 2 to perform as stated in page 40, lines

14 et seq.; and the means for running the finalization methods of any identified objects on the main thread prior to reset of the first heap [Id.].

Claim 34 relates to a computer system [FIG. 1, item 10] providing an object-based virtual machine [FIG. 1, item 40] environment for running successive applications. The computer system comprises storage [FIG. 5, item 560], at least a portion of which is logically divided into two or more heaps [FIG. 5, items 510 and 520] in which objects [FIG. 2, items 145] can be stored, wherein a first heap [FIG. 5, item 520] is reset between successive applications, and a second heap [FIG. 5, item 510] persists from one application to the next. A card table [FIG. 5, item 536] comprises multiple cards [paragraph 136], each corresponding to a region of said storage, each card in the card table being set to null when the first heap is reset between successive applications [page 6, lines 7-8 and page 59, lines 10-25];

logic for marking a card whenever an object in its corresponding storage region is updated [page 6, lines 9-10; page 59, lines 10-13];

logic for detecting possible references from the second heap to the first heap at reset by scanning the cards in the card table corresponding to the second heap, and detecting any cards which have been marked [page 6, lines 11-14; page 60, lines 3-16];

logic for locating, for each marked card, any objects in the corresponding region of storage [page 59, lines 3-9]; and logic for identifying any references to the first heap in the located objects [page 60, lines 3-16];

logic responsive to the identification of references to the first heap for performing the mark phase of a garbage collection to determine live objects in at least the second heap [page 61, lines 6-15]; logic for detecting whether any objects in the second heap having references to the first heap have been marked as live [page 61, lines 8-11]; and logic responsive to a detection of any such objects for returning an error condition to prevent reset for another application [page 7, lines 12-17];

logic for invalidating the card table if a compact operation has been performed on the second heap since the last reset [page 48, line 23-page 49, line 6];

logic for detecting references or possible references to the first heap from a set of predetermined locations [page 8, lines 16-25] ; and logic responsive to the detection of any such

references or possible references for returning an error condition to prevent reset for another application [page 56, lines 2-11];

storage, at least a portion of which is logically divided into two or more heaps in which objects can be stored, wherein a first heap is reset between successive applications, and a second heap persists from one application to the next [page 27, lines 12 et seq.];

logic for identifying any objects on the first heap which have a finalization method [page 40, lines 14-17; page 55, lines line 2-29]; and

logic for running the finalization methods of any identified objects on the main thread prior to reset of the first heap [page 9, lines 20-22; page 55, line 5-20];

wherein an object is only considered as within the region of storage corresponding to a card if a predetermined part of the object is in that region; and wherein the region of memory corresponding to a card comprises between 256 and 2048 bytes [page 59, lines 3-9]; and wherein the logic for performing the mark phase is also responsive to invalidation of the card table [page 61, lines 6-15].

**(6) Grounds of Rejection to be Reviewed on Appeal**

The grounds of rejection to be reviewed on appeal is:

1. Did the examiner properly reject claims 1-34 under U.S.C. §101 as being directed to non-statutory subject matter?
2. Did the examiner properly reject claims 1-34 under U.S.C. §103(a) as unpatentable over Wolczko (US Patent No. 5,900,001) in view of Aman US Patent No. 6,694,346) et al.?

**(7) Argument**

**(a) The examiner erred in rejecting claims 1-34 under U.S.C. §101**

The Office Action has rejected claims 1-11 and 23-34 under 35 U.S.C. §101 as being directed to non-statutory subject matter. The examiner said: "The claimed steps do not define a machine or computer implemented process." See Page 2 of the final Office Action. This is an error of law. To establish non-patentability of subject matter the Examiner must show that the claimed subject matter falls into one of the recognized exclusions from patentability. Excluded

from patent protection are laws of nature, physical phenomena, and abstract ideas. Parker v. Flook, 437 U.S. 584, 198 USPQ 193 (1978). He has not shown that the claimed invention falls into any of those categories. As in the recent decision, *Ex parte Lundgren*, 76 USPQ2d 1385 (BdPatApp&Int 2005), where the Board found that the so-called "non-technological art" exception did not fall into any of the excluded categories, the present grounds of "not embodied in a manner so as to be executable" fails. The Examiner's analysis is obviously flawed because claims 1-11 all claim computer systems which fall under the machine category of patentable subject matter. The elements in machine claims are not steps and computer systems are inherently executable. Claims 23-33 relate to a computer program product. The subject matter patentability of these claims as article of manufacture claims was recognized in *In re Beauregard*, 53 F.3d 1583 (Fed. Cir 1995). Without question, software code alone qualifies as an invention eligible for patenting under these categories, at least as processes. *Eolas Technologies Inc. v. Microsoft Corp.,.*, 73 USPQ2d 1782 (Fed. Cir. 2005), citing *In re Alappat*, 33 F.3d 1526 [31 USPQ2d 1545] (Fed. Cir. 1994); *AT&T Corp. v. Excel Communications, Inc.*, 172 F.3d 1352 [50 USPQ2d 1447] (Fed. Cir. 1999). Therefore, the rejection should be reversed.

The Examiner rejected claims 12-22 on grounds that the claimed invention is directed to non-statutory subject matter and that "[a] method is not tangibly embodied in a manner so as to be executable" and that "each of the claimed steps ...can be practiced mentally in conjunction with a pen and paper." That is incorrect. Take claim 12 for example; the preamble recites a method for operating a computer system and the elements are all steps for performing various manipulations of a card table. A card table is defined in the specification to be a data structure in a computer. See Paragraphs 136-137. It is absurd for the Examiner to suggest that these operations can be performed with a pen and paper.

Claim 21 is directed to the statutory category of "methods" and claim 23 is directed to the statutory category of "manufacture." Computer programs and machines implementing those programs have been held to constitute patentable subject matter. Section 101 explains that an invention includes "any new and useful process, machine, manufacture or composition of matter." 35 U.S.C. §101 (2000). The patented invention in this case is such a software product.

Thus, this software code claimed in conjunction with a physical structure fits within at least those two categories of subject matter within the broad statutory label of "patented invention." Therefore, this rejection should be reversed.

**(b) The examiner erred in rejecting claims 1-34 under 35 U.S.C. §103**

The Office Action rejected claims 1-34 as unpatentable over Wolczko (U.S. Patent 5,900,001) in view of Aman (U.S. Patent 6,694,346). Appellants contend that this is an error and should be reversed.

The Wolzco patent describes locating an object within a marked card but is being used by the examiner to open up the whole area of generational garbage collection as prior art. While the claimed approach is based on the techniques of generational garbage collection to identify objects in the persistent heap which have been modified to refer to objects in the resettable heap, the actual teaching in the Wolzco patent (locating an object within a marked card ) is not relevant to the instant patent application. Wolzco is prior to the Aman patent and Aman et al (engineers at IBM) are certainly skilled in the art of garbage collection and transaction processing. If it would have been obvious that it would be necessary to employ a card marking technique and further that the card table could be reset when the heap was reset then surely that should have been noted in Aman. Similarly if it would have been obvious that the finalizers should be run on objects in the resettable heap that should also have been noted. The Fact that Aman does not even discuss these concepts is actually evidence on their non-obviousness.

A claimed invention is unpatentable if the differences between it and the prior art "are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art." 35 U.S.C. 103(a) (Supp. 1998); *see Graham v. John Deere Co.*, 383 U.S. 1, 14, 148 USPQ 459, 465 (1966). The ultimate determination of whether an invention is or is not obvious is a legal conclusion based on underlying factual inquiries including: (1) the scope and content of the prior art; (2) the level of ordinary skill in the prior art; (3) the differences between the claimed invention and the prior art; and (4) objective evidence of nonobviousness. *See Graham*, 383 U.S. at 17-18, 148 USPQ at 467; *Miles Labs, Inc. v. Shandon Inc.*, 997 F.2d 870, 877, 27 USPQ2d 1123, 1128 (Fed. Cir. 1993).

The obviousness analysis begins in the text of section 103 quoted above, with the phrase "at the time the invention was made." For it is this phrase that guards against entry into the "tempting but forbidden zone of hindsight," *see Loctite Corp. v. Ultraseal Ltd.*, 781 F.2d 861, 873, 228 USPQ 90, 98 (Fed. Cir. 1985), overruled on other grounds by *Nobelpharma AB v. Implant Innovations, Inc.*, 141 F.3d 1059, 46 USPQ2d 1097 (Fed. Cir. 1998), when analyzing the patentability of claims pursuant to that section. Measuring a claimed invention against the standard established by section 103 requires the oft-difficult but critical step of casting the mind back to the time of invention, to consider the thinking of one of ordinary skill in the art, guided only by the prior art references and the then-accepted wisdom in the field. *See, e.g., W.L. Gore & Assoc., Inc. v. Garlock, Inc.*, 721 F.2d 1540, 1553, 220 UPSQ 303, 313 (Fed. Cir. 1983).

The best defense against the subtle but powerful attraction of a hindsight-based obviousness analysis is rigorous application of the requirement for a showing of the teaching or motivation to combine prior art references. *See, e.g., C.R. Bard, Inc. v. M3 Sys., Inc.*, 157 F.3d 1340, 1352, 48 USPQ2d 1225, 1232 (Fed. Cir. 1998) (describing "teaching or suggestion or motivation [to combine]" as an "essential evidentiary component of an obviousness holding"); *In re Rouffet,* 149 F.3d 1350, 1359, 47 USPQ2d 1453, 1459 (Fed. Cir. 1998) ("the Board must identify specifically . . . the reasons one of ordinary skill in the art would have been motivated to select the references and combine them"); *In re Fritch*, 972 F.2d 1260, 1265, 23 USPQ2d 1780, 1783 (Fed. Cir. 1992) (the examiner can satisfy burden of obviousness in light of combination "only by showing some objective teaching [leading to the combination]"); *In re Fine*, 837 F.2d 1071, 1075, 5 USPQ2d 1596, 1600 (Fed. Cir. 1988) (evidence of teaching or suggestion "essential" to avoid hindsight); *Ashland Oil, Inc. v. Delta Resins & Refractories, Inc.*, 776 F.2d 281, 297, 227 USPQ 657, 667 (Fed. Cir. 1985) (district court's conclusion of obviousness was in error when it "did not elucidate any factual teachings, suggestions or incentives from this prior art that showed the propriety of combination"). *See also Graham*, 383 U.S. at 18, 148 USPQ at 467 ("strict observance" of factual predicates to obviousness conclusion required). Combining prior art references without evidence of such a suggestion, teaching, or motivation simply takes the inventor's disclosure as a blueprint for piecing together the prior art to defeat patentability--the essence of hindsight. *See, e.g., Interconnect Planning Corp. v. Feil*, 774 F.2d 1132, 1138,

227 USPQ 543, 547 (Fed. Cir. 1985) ("The invention must be viewed not with the blueprint drawn by the inventor, but in the state of the art that existed at the time."). In this case, the Board fell into the hindsight trap.

The range of sources of obviousness available does not diminish the requirement for actual evidence in support of combining prior art: the showing must be clear and particular. *See*, *C.R. Bard, Inc., v. M3 Systems, Inc.*, 157 F.3d 1340, 1352, 48 USPQ2d 1225, 1232 (Fed. Cir. 1998). Broad conclusory statements regarding the teaching of multiple references, standing alone, are not "evidence." *E.g., McElmurry v. Arkansas Power & Light Co.*, 995 F.2d 1576, 1578, 27 USPQ2d 1129, 1131 (Fed. Cir. 1993) ("Mere denials and conclusory statements, however, are not sufficient to establish a genuine issue of material fact."); *In re Sichert*, 566 F.2d 1154, 1164, 196 USPQ 209, 217 (CCPA 1977) ("The examiner's conclusory statement that the specification does not teach the best mode of using the invention is unaccompanied by evidence or reasoning and is entirely inadequate to support the rejection."). In addition to demonstrating the propriety of an obviousness analysis, particular factual findings regarding the suggestion, teaching, or motivation to combine serve a number of important purposes, including: (1) clear explication of the position adopted by the Examiner and the Board; (2) identification of the factual disputes, if any, between the Appellant and the Board; and (3) facilitation of review on appeal. Here, however, the Office Action did not make particular findings regarding the suggestion, teaching, or motivation to combine the prior art references. Instead the Examiner contends that it would have been obvious to combine the references because "it would improve Wolczko's system by allowing each heap to be cleared of application memory thus improving the overall system as taught by Aman." That statement mischaracterizes the claim language and only attacks a straw man. Take claim 1 as an example, it specifies two or more heaps and only one heap is cleared (the first heap). The other heap (the second heap) is not cleared at all. The examiner has clearly not understood this aspect of the invention.

Claim 1 recites a computer system providing an object-based virtual machine environment for running successive applications, the computer system including storage, at least a portion of which is logically divided into two or more heaps in which objects can be stored,

wherein a first heap is reset between successive applications, and a second heap persists from one application to the next, the system including: a card table comprising multiple cards, each corresponding to a region of said storage, each card in the card table being set to null when the first heap is reset between successive applications; logic for marking a card whenever an object in its corresponding storage region is updated; and logic for detecting possible references from the second heap to the first heap at reset by scanning the cards in the card table corresponding to the second heap, and detecting any cards which have been marked.

The Wolczko patent actually describes a way to locate an object in a card and relies on tagged storage to do it. It only contains a description of known card marking technique which is but a single element of the combination claimed herein. The Aman patent describes the technique of clearing the transient heap after a transaction and that is another element of the claimed invention. The Examiner concedes that Wolczko does not disclose the use of resetting heap applications. However, the Examiner contends that Aman teaches this element and that it would have been obvious to one skilled in the art to modify Wolczko according to Aman. Appellant respectfully submits that claim 1 would not have been obvious in view of the combination of Wolczko and Aman. We claim that the card table is set to null when a reset occurs. That limitation is neither taught, motivated, nor suggested by the combination of Wolczko and Aman. Aman does not even discuss card tables. Wolczko discusses card tables but as admitted by the Examiner does not teach resetting the heap at all. There is no evidence anywhere of any suggestion to combine these references. Proving obviousness must be more than merely finding elements in the prior art and piecing some of the general teachings together without any specific suggestion to do so.

The Examiner sustained his rejection under 35 U.S.C. §103 arguing that "Aman [sic: Aman's] method of resetting memory heaps after applications would improve Wolczko's system by allowing each heap to be cleared of previous application memory thus improving the overall system as taught by Aman." If achieving an "improvement" were motivation then very few inventions would be patentable but that is not the law. The Examiner used an insufficient reason for combining the references. Rather the Examiner really bases the combination on Appellant's own teachings. "Defining the problem in terms of its solution reveals improper hindsight in the

selection of the prior art relevant to obviousness." *Pro-Mold & Tool Co. v. Great Lakes Plastics, Inc.,* 75 F.3d 1568, 1573, 37 USPQ2d 1626, 1630 (Fed. Cir. 1996). The problem confronted by the inventor must be considered in determining whether it would have been obvious to combine references in order to solve the problem. *Northern Telecom, Inc. v. Dataflow,* 908 F.2d 931 (Fed. Cir. 1990); *Diversitech Corp. v. Century Steps, Inc.*, 850 F.2d 675 (Fed. Cir. 1988). Appellant's use of the card table as specified in claim 1 is done because of a problem recognized by Appellants, not the prior art. Thus the specification provides:

"One complication (not shown in FIG. 10) is that promoting an object from the transient heap to the middleware heap may lead to an allocation failure on the middleware heap if space is exhausted. In such an eventuality, a garbage collection is performed. If this still does not create enough space, then this will lead to error 1099." Paragraph 135.

The use of the card table solves the problem. Neither Wolczko nor Aman would have motivated that solution. Neither of those references even dealt with the above problem.

Claims 2-9 are dependent on claim 1 and hence would not have been rendered obvious by the cited references for at least the same reasons as claim 1. Claim 12 is a method counterpart of claim 1 and would not have been rendered obvious by the cited references for at least the same reasons as claim 1. Claims 13-20 are dependent on claim 12 and hence would not have been rendered obvious by the cited references for at least the same reasons as claim 1. Claim 23 is a computer program product counterpart of claim 1 and it and its dependent claims (24-31) are dependent on claim 1 and hence would not have been rendered obvious by the cited references for at least the same reasons as claim 1.

Claim 10 relates to a computer system providing an object-based virtual machine environment for running successive applications, said computer system comprising: storage, at least a portion of which is logically divided into two or more heaps in which objects can be stored, wherein a first heap is reset between successive applications, and a second heap persists from one application to the next means for identifying any objects on the first heap which have a finalization method; and means for running the finalization methods of any identified objects on the main thread prior to reset of the first heap. Neither Wolczko nor Aman teach or suggest the invention as claimed. Specifically, neither reference teaches a pair of heaps where one heap is

reset between applications and the other persists from one application to the next. In the rejection of claim 1 (which included this limitation) the Examiner conceded that " "Wolczko does not specifically teach the use of resetting [sic, the] heap at applications." However, now the Examiner apparently contends that it does. See page 6 of the final rejection. That is an error that should be reversed. The cited references also fail to teach or suggest identifying any objects on the first heap which have a finalization method. The Examiner cites Aman, col. 4, lines 1-16 in support of this rejection. It is worthwhile to quote that section to show that it had nothing to do with finalization methods:

"Resettable portion 32 is used to store application specific code, whereas persistent portion 34 stores static values, i.e., predefined values used to bring an application program to an initial state. In similar fashion, application 24 is associated with private heap 36 comprising a resettable portion 38 and a persistent portion 40.

Also resident in memory 18 is a shared heap 42 which includes a read-only portion 44 and a read/write portion 46. Read-only portion 44 is used to store non-application specific code (e.g., run-time classes) and read/write portion provides an area of memory that either application 22 or 24 can read from or write to, using suitable lock protection to avoid conflicts. Private heap 30 can only be accessed by application 22, and private heap 36 by application 24, however, shared heap 42 is accessible by both applications 22 and 24."

There is no mention of detecting any objects in the first heap which have a *finalization* method. The Aman approach is quite different from that of claim 10. Instead of having two heaps, as claimed, Aman has a shared heap and a private heap, each having a persistent portion and a resettable portion. That is not the same as two heaps, one resetting from application to application and the other being persistent. Moreover, the claimed heaps are not associated with a single application (note the reference to applications).

Claim 11 is dependent on claim 10 and hence would not have been rendered obvious by the cited references for at least the same reasons as claim 10. Claim 21 is a method counterpart of claim 10 and it and its dependent claims would not have been rendered obvious by the cited references for at least the same reasons as claim 10. Claim 32 is a computer program product counterpart of claim 10 and hence would not have been rendered obvious by the cited references for at least the same reasons as claim 10.

Claim 34 is not unpatentable over the cited references for the reasons discussed above and in addition for the following reasons. The combination of references does not teach or

suggest the logic for detecting possible references from the second heap to the first because the references do not teach or suggest first and second heaps. Aman at 53-65 speaks of several applications sharing a private heap and using a part of this shared heap that is not reset. The claimed invention uses a separate heap that is not reset from application to application and a second heap that is reset. The claimed heaps are either resettable or persist, neither of them is divided into resettable portions and persistent portions. Such an interpretation of the claims is inconsistent with their purpose and would not be an appropriate claim interpretation.

The cited references also fail to teach or suggest the claimed logic for locating, for each marked card, any objects in the corresponding region of storage; and logic for identifying any references to the first heap in the located objects; logic responsive to the identification of references to the first heap for performing the mark phase of a garbage collection to determine live objects in at least the second heap; logic for detecting whether any objects in the second heap having references to the first heap have been marked as live; and logic responsive to a detection of any such objects for returning an error condition to prevent reset for another application. The office action cites Wolczko at col. 5, lines 2-30; col. 6, lines 1-11; col. 14, lines 21-52; col. 20, lines 33-60.

Wolczko, Col. 5, lines 2-30 reads:

"One optimization used in the prior art is to segment the heap into equal size areas (called cards) and to mark each card when a write operation occurs within the card--a form of a write-barrier. Thus, only cards marked as `dirty` (instead of all the cards in the heap memory) are searched for pointers when updating the root set. FIG. 1b illustrates the use of card marking. A general reference character 120 illustrates a card-marked region of memory 121. The card-marked region of memory 121 contains a first card 123 and a second card 125. In this illustration, the first card 123 is adjacent in memory to the second card 125. Thus a plurality of nodes (A-F) 127 are distributed over the first card 123 and the second card 125. The first card 123 is associated with a first card marker 129 and the second card 125 is associated with a second card marker 131. When memory is modified in one of the cards 123, 125, the appropriate card marker is flagged. Thus, in the illustration of FIG. 1b, a write operation was performed within the first card 123 resulting in the first card marker 129 being marked `dirty` as indicated by the `X` in the first card marker 129. The fact that the second card marker 131 is not marked indicates that none of the memory in the second card 125 has been modified since the last scavenge. The fact that a node `D` 133 extends across the boundary between the first card 123 and the second card 125 complicates the ability to detect the start of the node. Generally, card markers are initialized to all ones (FF hex) because the computer's memory-clear operation is often faster than a store-value operation."

Wolczko, Col. 6, lines 1-11 read:

"Another problem with cardmarking is that the operation of scanning the card indicators to find the marked cards is an overhead operation because a large number of memory locations (those containing the marking vector) must be examined to locate the marked cards."

Wolczko, Col. 14, lines 21-52 reads:

"Another problem with cardmarking is that the operation of scanning the card indicators to find the marked cards is an overhead operation because a large number of memory locations (those containing the marking vector) must be examined to locate the marked cards."

Wolczko, Col. 20, lines 33-60 reads:

"FIG. 4c illustrates a routine dispatch table as indicated by general reference character 450. The routine dispatch table 450 contains pointers, handles, or similar means to invoke called routines. When invoked, each called routine receives, as an argument, a pointer to the current variable in the instance variable storage area 403. A first entry 451 in the routine dispatch table 450 contains a pointer to a procedure that does not process any instance variables in the instance variable storage area 403. The first entry 451 corresponds to an entry in the tagging bitmap 430 that indicates no pointers in the next eight variables of the instance variable storage area 403. A second entry 453 in the routine dispatch table 450 contains a pointer to a called routine to process INDEX(0) 455. This called routine 455 processes only the variable in the instance variable storage area 403 pointed to by the passed argument. A third entry 457 in the routine dispatch table 450 contains a pointer to a called routine to process INDEX(3) 459 that only processes the third instance variable beyond the one pointed to by the passed argument. A fourth entry 461 in the routine dispatch table 450 contains a pointer to a called routine to process INDEX(O) and INDEX(6) 463. This routine 463 processes both the variable pointed to by the passed argument and the sixth variable past the variable pointed to by the passed argument. Finally, a fifth entry 465 in the routine dispatch table 450 contains a pointer to a called routine to process INDEX(O) through INDEX(7) 467. This routine 467 processes all eight instance variables starting at the instance variable pointed to by the passed argument. Thus, each called routine processes a pattern of pointer and non-pointer instance variables starting at the passed argument."

Nowhere in the sections cited above are there any of the claim limitations for which they are cited.

The cited combination of references also fails to teach or suggest the logic for detecting references from the first heap to the second heap because the cited references do not teach first and second heaps as claimed. See the discussion of this element above.

The cited combination of references also fails to teach or suggest the claimed logic responsive to the detection of any such references or possible references for returning an error

condition to prevent reset for another application. The cited portion of Wolczko says nothing about an error condition at all.

The cited combination of references also fails to teach or suggest the claimed logic for running the finalization methods of any identified objects on the main thread prior to reset of the first heap. In Wolczko col. 4, lines 1-20, the discussion is about a single heap, not two as claimed. The same is true about col. 3, lines 59-67. Similarly, col. 2, lines 556-67 say nothing at all about the claimed first and second heaps. Col. 5, lines 1-5 discussed cards in a very general sense and does not come even close to the use of cards according to the invention.

The cited combination of references also fails to teach or suggest the claimed region of memory corresponding to a card comprising between 256 and 2048 bytes. That size of region is not even hinted at in the cited references at all.

For the foregoing reasons Appellant requests reversal of the rejections and allowance of the claims.

Please apply the brief fee of $500 and any other charges or credits to Deposit Account No. 50-0510.

Respectfully submitted,

Date: January 4, 2006

Michael J. Buchenhorner

Michael J. Buchenhorner
Reg. No. 33,162

Holland & Knight LLP
701 Brickell Avenue
Miami, Florida
Telephone: (305) 789-7773
Facsimile: (305) 789-7799

## Appendix of Claims

Claim 1. A computer system providing an object-based virtual machine environment for running successive applications, said computer system comprising:

storage, at least a portion of which is logically divided into two or more heaps in which objects can be stored, wherein a first heap is reset between successive applications, and a second heap persists from one application to the next;

a card table comprising multiple cards, each corresponding to a region of said storage, each card in the card table being set to null when the first heap is reset between successive applications;

logic for marking a card whenever an object in its corresponding storage region is updated; and

logic for detecting possible references from the second heap to the first heap at reset by scanning the cards in the card table corresponding to the second heap, and detecting any cards which have been marked.

Claim 2. The computer system of claim 1, further comprising: logic for locating, for each marked card, any objects in the corresponding region of storage; and logic for identifying any references to the first heap in the located objects.

Claim 3. The computer system of claim 2, further comprising: logic responsive to the identification of references to the first heap for performing the mark phase of a garbage collection to determine live objects in at least the second heap; logic for detecting whether any objects in the second heap having references to the first heap have been marked as live; and logic responsive to a detection of any such objects for returning an error condition to prevent reset for another application.

Claim 4. The computer system of claim 3, further comprising logic for invalidating the card table if a compact operation has been performed on the second heap since the last reset, wherein said means for performing the mark phase is also responsive to invalidation of the card table.

Claim 5. The computer system of claim 1, wherein an object is only considered as within the region of storage corresponding to a card if a predetermined part of the object is in that region.

Claim 6. The computer system of claim 1, wherein the region of memory corresponding to a card comprises between 256 and 2048 bytes.

Claim 7. The computer system of claim 1, further comprising: logic for detecting references or possible references to the first heap from a set of predetermined locations; and logic responsive to the detection of any such references or possible references for returning an error condition to prevent reset for another application.

Claim 8. The computer system of claim 7, wherein the set of predetermined locations includes the stacks and registers.

Claim 9. The computer system of claim 1, further comprising: logic for detecting any objects on the first heap which are reachable from virtual machine system class objects; and logic for promoting any such detected objects to the second heap.

Claim 10. A computer system providing an object-based virtual machine environment for running successive applications, said computer system comprising:

storage, at least a portion of which is logically divided into two or more heaps in which objects can be stored, wherein a first heap is reset between successive applications, and a second heap persists from one application to the next

means for identifying any objects on the first heap which have a finalization method; and

means for running the finalization methods of any identified objects on the main thread prior to reset of the first heap.

Claim 11. The computer system of claim 10, further comprising logic responsive to running the finalization methods for checking that they have not performed any operations which would prevent reset of the first heap.

Claim 12. A method of operating a computer system providing an object-based virtual machine environment for running successive applications, the computer system including storage, at least a portion of which is logically divided into two or more heaps in which objects can be stored, wherein a first heap is reset between successive applications, and a second heap persists from one application to the next, said method including the steps of:

providing a card table comprising multiple cards, each corresponding to a region of the storage, each card in the card table being set to null when the first heap is reset between successive applications;

marking a card whenever an object in its corresponding storage region is updated; and

detecting possible references from the second heap to the first heap at reset by scanning the cards in the card table corresponding to the second heap, and detecting any cards which have been marked.

Claim 13. The method of claim 12, further comprising: locating, for each marked card, any objects in the corresponding region of storage; and identifying any references to the first heap in the located objects.

Claim 14. The method of claim 13, further comprising: responsive to the identification of references to the first heap, performing the mark phase of a garbage collection to determine live objects in at least the second heap; detecting whether any objects in the second heap having references to the first heap have been marked as live; and responsive to a detection of any such objects, returning an error condition to prevent reset for another application.

Claim 15. The method of claim 14, further comprising the step of invalidating the card table if a compact operation has been performed on the second heap since the last reset, wherein said step of performing the mark phase is also performed in response to invalidation of the card table.

Claim 16. The method of claim 12, wherein an object is only considered as within the region of storage corresponding to a card if a predetermined part of the object is in that region.

Claim 17. The method of claim 12, wherein the region of memory corresponding to a card comprises between 256 and 2048 bytes.

Claim 18. The method of claim 12, further comprising: detecting references or possible references to the first heap from a set of predetermined locations; and responsive to the detection of any such references or possible references, returning an error condition to prevent reset for another application.

Claim 19. The method of claim 18, wherein said set of predetermined locations includes the stacks and registers.

Claim 20. The method of claim 12, further comprising: detecting any objects on the first heap which are reachable from virtual machine system class objects; and promoting any such detected objects to the second heap.

Claim 21. A method of operating a computer system providing an object-based virtual machine environment for running successive applications, said computer system including storage, at least a portion of which is logically divided into two or more heaps in which objects can be stored, wherein a first heap is reset between successive applications, and a second heap persists from one application to the next, said method including the steps of: identifying any objects on the first heap which have a finalization method; and running the finalization methods of any identified objects on the main thread prior to reset of the first heap.

Claim 22. The method of claim 21, further comprising the step, responsive to running said finalization methods, of checking that they have not performed any operations which would prevent reset of the first heap.

Claim 23. A computer program product for operating a computer system providing an object-based virtual machine environment for running successive applications, the computer system including storage, at least a portion of which is logically divided into two or more heaps in which objects can be stored, wherein a first heap is reset between successive applications, and a second heap persists from one application to the next, said computer program product comprising machine-readable program instructions recorded on a storage medium, the instructions when loaded into the computer system causing it to perform the steps of: providing a card table comprising multiple cards, each corresponding to a region of the storage, each card in the card table being set to null when the first heap is reset between successive applications; marking a card whenever an object in its corresponding storage region is updated; and detecting possible references from the second heap to the first heap at reset by scanning the cards in the card table corresponding to the second heap, and detecting any cards which have been marked.

Claim 24. The computer program product of claim 23, wherein said program instructions further cause the computer to perform the steps of: locating, for each marked card, any objects in the corresponding region of storage; and identifying any references to the first heap in the located objects.

Claim 25. The computer program product of claim 24, wherein said program instructions further cause the computer to perform the steps of: responsive to the identification of references to the first heap, performing the mark phase of a garbage collection to determine live objects in at least the second heap; detecting whether any objects in the second heap having references to the first heap have been marked as live; and responsive to a detection of any such objects, returning an error condition to prevent reset for another application.

Claim 26. The computer program product of claim 25, wherein said program instructions further cause the computer to perform the step of invalidating the card table if a compact operation has been performed on the second heap since the last reset, wherein said step of performing the mark phase is also performed in response to invalidation of the card table.

Claim 27. The computer program product of claim 23, wherein an object is only considered as within the region of storage corresponding to a card if a predetermined part of the object is in that region.

Claim 28. The computer program product of claim 23, wherein the region of memory corresponding to a card comprises between 256 and 2048 bytes.

Claim 29. The computer program product of claim 23, wherein said program instructions further cause the computer to perform the steps of: detecting references or possible references to the first heap from a set of predetermined locations; and responsive to the detection of any such references or possible references, returning an error condition to prevent reset for another application.

Claim 30. The computer program product of claim 29, wherein said set of predetermined locations includes the stacks and registers.

Claim 31. The computer program product of claim 23, wherein said program instructions further cause the computer to perform the steps of: detecting any objects on the first heap which are reachable from virtual machine system class objects; and promoting any such detected objects to the second heap.

Claim 32. A computer program product for operating a computer system providing an object-based virtual machine environment for running successive applications, said computer system including storage, at least a portion of which is logically divided into two or more heaps in which objects can be stored, wherein a first heap is reset between successive applications, and a second heap persists from one application to the next, said computer program product comprising machine-readable program instructions recorded on a storage medium, said instructions when loaded into the computer system causing it to perform the steps of: identifying

any objects on the first heap which have a finalization method; and running the finalization methods of any identified objects on the main thread prior to reset of the first heap.

Claim 33. The computer program product of claim 21, wherein said program instructions further cause the computer to perform the step responsive to running said finalization methods, checking that they have not performed any operations which would prevent reset of the first heap.

Claim 34. A computer system providing an object-based virtual machine environment for running successive applications, said computer system comprising:

storage, at least a portion of which is logically divided into two or more heaps in which objects can be stored, wherein a first heap is reset between successive applications, and a second heap persists from one application to the next;

a card table comprising multiple cards, each corresponding to a region of said storage, each card in the card table being set to null when the first heap is reset between successive applications;

logic for marking a card whenever an object in its corresponding storage region is updated;

logic for detecting possible references from the second heap to the first heap at reset by scanning the cards in the card table corresponding to the second heap, and detecting any cards which have been marked;

logic for locating, for each marked card, any objects in the corresponding region of storage; and logic for identifying any references to the first heap in the located objects;

logic responsive to the identification of references to the first heap for performing the mark phase of a garbage collection to determine live objects in at least the second heap; logic for detecting whether any objects in the second heap having references to the first heap have been marked as live; and logic responsive to a detection of any such objects for returning an error condition to prevent reset for another application;

logic for invalidating the card table if a compact operation has been performed on the second heap since the last reset;

logic for detecting references or possible references to the first heap from a set of predetermined locations; and logic responsive to the detection of any such references or possible references for returning an error condition to prevent reset for another application;

storage, at least a portion of which is logically divided into two or more heaps in which objects can be stored, wherein a first heap is reset between successive applications, and a second heap persists from one application to the next;

logic for identifying any objects on the first heap which have a finalization method; and

logic for running the finalization methods of any identified objects on the main thread prior to reset of the first heap;

wherein an object is only considered as within the region of storage corresponding to a card if a predetermined part of the object is in that region; and wherein the region of memory corresponding to a card comprises between 256 and 2048 bytes; and wherein the logic for performing the mark phase is also responsive to invalidation of the card table.